Dynamic Effective Precision Matching Computation

Goulart, Victor
Department of Informatics, Kyushu University

Murakami, Kazuaki
Department of Informatics, Kyushu University

出版情報：The 11th Workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI 2003), pp.230-237, 2003-04. Sasimi Workshop
バージョン：
権利関係：

Dynamic Effective Precision Matching Computation

Victor Goulart and Kazuaki Murakami
Department of Informatics
Kyushu University
Kasuga-koen 6-1, Kasuga, Fukuoka, Japan 816-8580
e-mail: *victor.goulart@acm.org*

Abstract — In this paper, the authors discuss fine-grain power savings obtained through a technique which we call Effective Precision Matching Computation—PreMatch. It works by dynamically selecting arithmetic modules in a system designed with Functional Redundancy. Functional Redundant (FR) systems are those which contain different implementation instances of functional blocks or circuits on chip. The different instances are obtained by using different hardware algorithms and/or precision (bitwidth) of their operands. The selection of modules is completely dynamic and data-driven, i.e., it is done according to the contents of the parameters passed as input to those modules. Basically, according to the values of the operands different instances of adders, multipliers, and so on can be selected to "match" the required precision of the computation. Experiments showed that up to 28.4% energy savings can be achieved with PreMatch alone, and up 29.3% using both techniques (FR and PreMatch).

Keywords: Low Power, Functional Redundancy, Module Selection, Bitwidth Optimization, Precision Matching.

I. Introduction

As power consumption becomes one of the big concerns of system design, low power techniques are urging, making low power one of the hottest research topics actually. The reason is that increased power consumption directly impacts CPU and system cost. After clock, the datapath is the largest power consuming component in a high-performance CPU [13]; datapath power is basically due to register file and functional units power (i.e. power consumed at arithmetic and logic operations). Focusing on these type of instructions and how to reduce power when executing them is an effective power reduction technique (specially because there is a lot of redundancy and useless power consumed when executing those instructions). We will be back to this topic later. One of the biggest challenges, though, is how to achieve great power savings with little or no performance impact at all.

One effective way to get power reductions is through datapath width adjustment. The basic approach works by carefully analyzing datapath width requirements for each application; designers can determine the length of registers, memory size and word lengths, and so on in order to minimize chip area and power consumption. Nevertheless, system programmers pay little attention to the necessary bitwidth of data-types used in their programs. For instance, it is common to find in C programs use of 32-bit integer data-type to represent a single Boolean variable; when operating on this variable the entire datapath was exercised regardless of operand size. In order to release the programmer from this burden some works have focused on compilers to minimize bitwidth [12]. The waste becomes worst as processor's datapath becomes wider. In [5] datapath analysis of MPEG-2 video decoder program was performed and showed that only 35% of the total bits of the 384 variables declared *int* type are actually used in the computation; about 50 variables are used as flags which requires only 1 bit.

As well as the static approach, normally used for ASICs design, it is possible to obtain great power savings by the dynamic recognition of operand values; discovering the effective precision of each computation and dynamically matching the datapath width and the latest precision. Power savings are obtained by stripping off unnecessary switching activity of the system. Other similar approaches are precomputation, signal gating, canonical signal digits, computation redundancy reduction, among others.

In this work we assume a processor which presents functional redundancy – such as functional units (or modules) implemented for different bitwidth data; those modules can even be implemented using different hardware algorithms for better exploration space on power reductions or performance improvements.

Fig. 1 shows the cumulative percentage of integer instructions in SPECint2000 in which both operands are less than or equal to the specified bitwidth in a 32-bit datapath processor. This data represents the effective bitwidths of operands being executed in the pipeline. Some benchmarks like `cc1` and `bzip2` present an interesting concentration of effective bitwidths on 0 to 8 bits and 16 to 21 bits respectively, showing the specificity of optimum

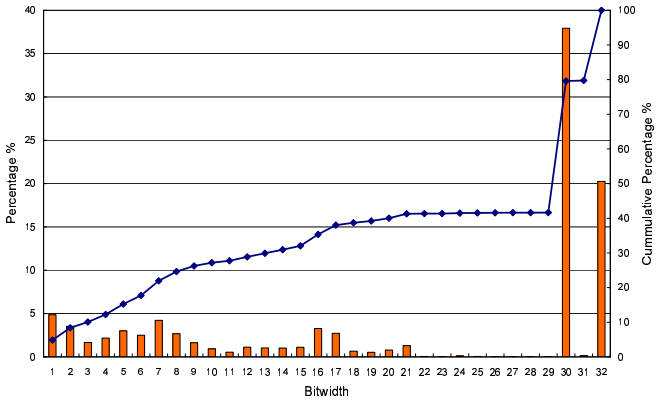

Fig. 1. Actual operand size (in bitwidths) of additions in SPECint2000 benchmarks.

datapath width according to the application being handled. All benchmarks present a peak on 30 and 32 bits values mainly due to address calculations.

In order to exploit effective narrow width operands (on-the-fly) one must dynamically recognize operand values; dynamically discover the effective precision necessary for the computation and dynamically perform the operation for the relative effective precision. In Brooks' work [3], power savings are obtained by clock gating unused parts of adders discovered by Zero-detect circuitry (narrow bitwidth operands). The granularity of this approach, i.e. how many bits may be clock gated, is heavily dependent on the structure of the adder, multiplier, etc.

In this paper we explore the multifariousness of FR systems not only with respect to their diverse functionality implementation but also to the bitwidth of the data is going to be manipulated by those modules. We propose a processor designed with functional redundant elements (like adders, multipliers, and so on) to reduce useless power consumptions due to over-precision calculations, i.e., performing computations with more bitwidth than is actually necessary. This implies the contents of the operators to select on-the-fly which algorithm and/or bitwidth will be appropriate to perform the computation. No compiler intervention is necessary and the module selection is completely dynamic and data-driven.

The rest of this paper is organized as follows: In Section II. presents the background on Functional Redundant (FR) systems. Section III. presents our proposed technique, called **Effective Precision Matching Computation**, and how it applies on FR systems. In Section IV. we present some experimental results. Sections V. and VI. discuss related work and concludes with some directions for future work.

II. Functional Redundancy Systems

In this section we present the Functional Redundancy [7] approach for system design. It is a design methodology which allows the dynamic exploitation of performance and power/energy consumption in IC systems. [7] is a conceptual paper on Functional Redundancy (FR) and on its potential as an effective system design methodology for low power systems with minimum performance degradation or none at all. It is heavily IP-oriented and shows to be an effective solution for dynamic exchange of system characteristics even after chip tape-out. This methodology is best suited for dynamic systems in which system flexibility, on-demand high performance and low power consumption is a much more important concern than die size.

Even though one can use a very powerful notebook running over 2GHz, all this computation power is not necessary to tasks as text editors, but may not be enough for rendering virtual-reality (VR) worlds or even multimedia applications. In our approach the selection of IP cores, among a set of cores with same functionality will be determined by the contents of the operands relative to each core or module. In the case of a text editor, low speed, low power consuming IP cores or modules should be used, while for VR environments high performance ones should be chosen for executing the task.

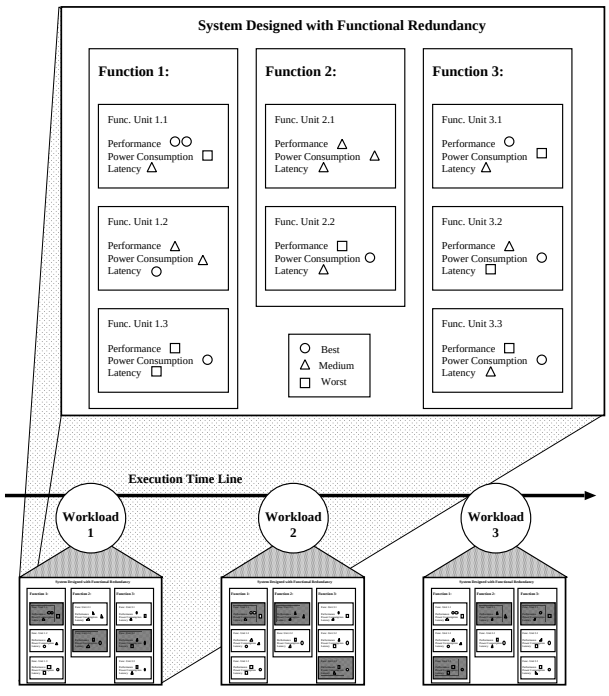

Fig. 2. Functional Redundancy: as the workload changes, different parts of the circuit are put active in order to deliver the required performance and meet timing constraints, minimizing power consumption and maximizing system's flexibility.

The basic idea is to have in the same circuit, different instances (IP cores) with same functionality, but with different performance-energy consumption characteristics. Power savings can be achieved with low or no performance lost by the appropriate selection of modules at execution time.

A general system with Functional Redundancy (three different cores for system function 1; two different cores for function 2; and three cores for function 3) is shown in Fig. 2. In a given moment, the system will be in one of all possible different configurations, in which one core of each function will be active. During task's execution the workload will change (Workload 1, Workload 2, and so on). According to the workload, available parallelism, QoS—Quality of Service parameters among others, different module selection policies can be applied in order to dynamically specify which cores will be used to execute the actual code section of the program.

By effectively selecting modules during program execution it is possible to obtain minimum power consumption, maximizing system's flexibility to adapt itself to the environment requirements. *Recompilation of code is not necessary as portions of circuit instead of instructions are scheduled during task's execution.* However some instruction/module matching and scheduling done at compile time can improve the performance and/or power consumption characteristics of the system.

Many different granularities of functional redundancy can exist: from FU (Functional Unit) till whole (set of) processors. It is much simpler to duplicate than share resources, and redundancy of resources is one of the core principles for reliable computing. In [11] redundancy is used at the pipeline level in a OOO (Out-of-Order) processor core. Two OOO pipelines are used: one running at a slower clock frequency and the other at the same clock frequency as the rest of the processor. The number of stages of the pipelines is different to compensate their different clock domains, which complicates the control logic.

In order to demonstrate the effects of algorithm selection, we have analyzed a number of different implementations of adders (Fig. 3). Big differences on power consumption, computation time, PDP (Power-Delay Product) can be observed not only between different adder algorithms but also according to the different data bitwidths or precisions involved. PDP$\times$Area shows the compromises with area, power and delay altogether.

III. Effective Precision Matching Computation

In this section we present **PreMatch—Effective Precision Matching Computation** technique for low power, fast computation. This technique explores the potential power savings which can be achieved using different circuit implementations and/or with different precisions.

We designed and analyzed power, delay, area and PDP (Power-Delay Product) of the following adders: Brent-Kung Adder [2], ELM Adder [9], Carry-Look-Ahead Adder [8], Carry-Skip Adder [14, 8], Carry-Select Adder [1, 8] and ELM Adder [10]. Their characteristics are showed on Fig. 3. One can easily observe that the characteristics of power and delay greatly vary according to the hardware algorithm used and to the precision of the operation (operand bitwidth).

Our preliminary study on the effective bitwidth of operands in SPECInt2000 (Fig. 1) showed that a great percentage of operations are low precision; the same was observed in [3] where over 50% of operations used less than 16 bits for the SPECInt95 in a 64-bit Alpha-like machine simulation.

As pointed out earlier, in order to explore narrow width operands we must recognize the operand values (this is done at the decode stage of the pipeline); dynamically discover the effective precision for the computation (this is done also at the decode stage when the contents of the register file are read and from the forward circuit of the result of a previous computation). For example, in case of a 8-bit register the values 20, +20 and -1 or *unsigned 00010100, signed 00010100, signed 11111111* have effective binary values of *10100, 010100* and *1* needing 5, 6 and 1 bits respectively. This analysis is only possible after instruction decode when it is possible to know if the most significant bit of a register is a sign bit or not. The final part is dynamically match the datapath width (this is done by multiplexing the operand values to the appropriate circuits, which has low overhead on circuit area and power for a small number of redundant cores). There is an import tradeoff here and it has to be analysed according to the application requirements of power, performance and its data bitwidth pattern (as in Fig. 1).

Differently from [3] we do not need to perform the zero-detection to discover the operands precison to all operands everytime. One could annotate the precision of an operand when it is analyzed at first time when it enters the pipeline and then update the results bitwidth according to the operation after execution. A simple implementation would be some flags at the register file to identify the precision of the operands and a small table inside the pipeline to identify result operands on-the-fly. The next time this operand is going to be used one just needs to check the other operands precision, case it has not being done before (an operand taken from memory).

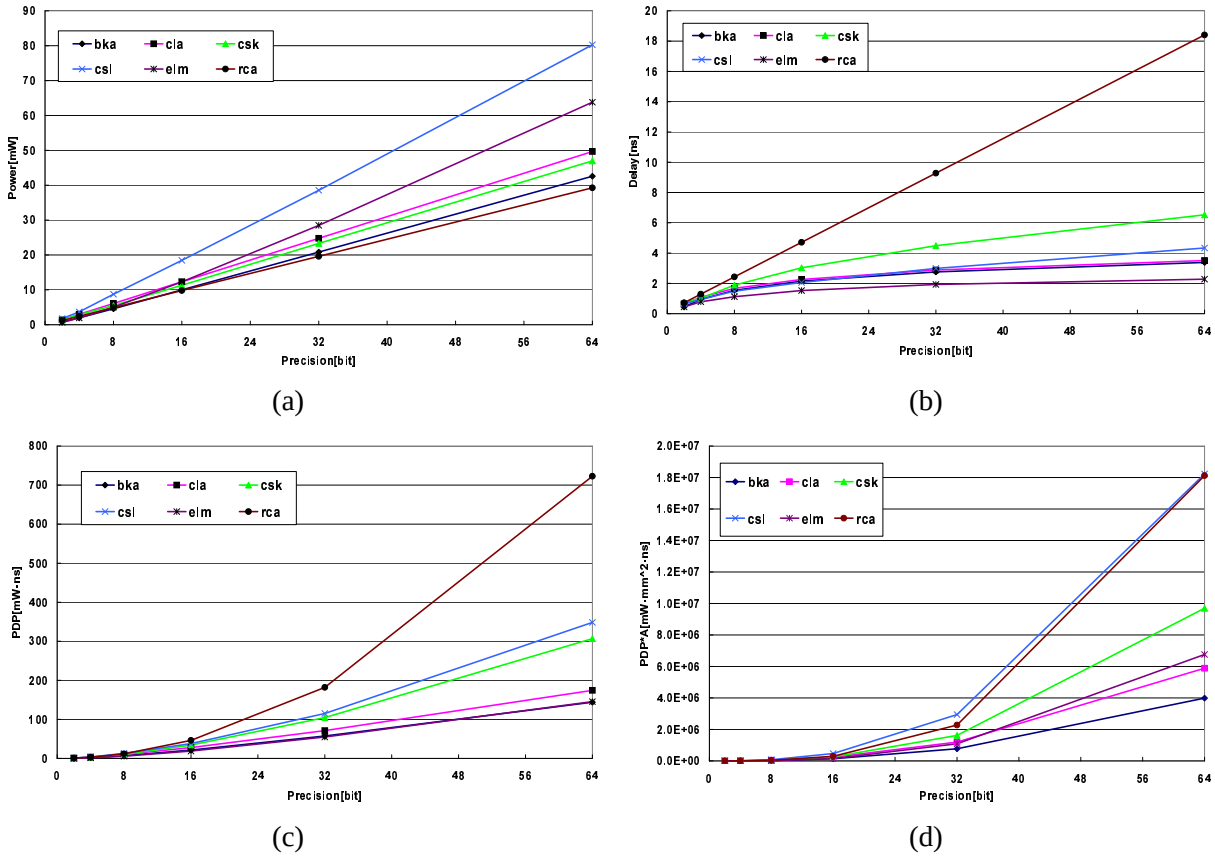

Fig. 3. (a) Power; (b) Delay; (c) Power-Delay Product (PDP); and (d) PDP$\times$Area for each adder.

IV. Evaluation

In this section we evaluate PreMatch—Effective Precision Matching Computation as a power saving technique. Addition is by far the most frequently used operation, whether the system is a general-purpose or application specific processor. Multiplication although not as present as addition consumes by far much more power. We evaluate PreMatch effectiveness for a class of adders with different bitwidths and characteristics of power-delay product. This can be easily extended to the study on multipliers or other more complex/power hungry functions.

Improvements on power consumption are compared against a base system. The base machine does not contain any functional redundancy with resources which operate at the full datapath width. The proposed machines contain functional redundancy in the form of PreMatch or by using different hardware algorithms. The types of machines studied are summarized in Table I. Three machine "families" are studied, high performance, high power consuming ELMA machines, low performance, very low power consuming RCA machines and the CSkA for medium to high performance and medium power consumption.

Machines type A are the reference machines because they just have adders for the full width of the datapath (32-bits). RCA$_A$ being the lowest power consuming, but with great operation latency and ELMA$_A$ being the most power consuming and high performance one, and CSkA$_A$ which are not so power hungry as ELMA machines nor as slow as RCA ones. In machines type B, C and H we apply PreMatch using adders for narrow operands of 8, 16 and 16-8-4-2 respectively. This is similar to the technique proposed in [3]; however machines like type H are not easily implementable as in some cases the complexity of the selection logic and number of connections might impact the performance of the FU. Machines type D, E, F, G and I explore both Functional Redundancy and PreMatch.

A. Experiments

In order to evaluate the power and performance characteristics of the various machines described in Table I, we use the Wattch toolset [4]. The parameters associated with power consumption of FU's (Table III) are scaled according to the values we obtained from our study over different adders characteristics in 0.18μm technology. All machines are OOO and the characteristics of this base processor are described in Table II.

TABLE I
MACHINE CONFIGURATIONS.

Name	Type
RCA_A	Base RCA_{32}
RCA_B, RCA_C	PreMatch (RCA_{32} + $RCA_{8,16}$)
RCA_D, RCA_E	FR-PreMatch (RCA_{32} + $CSkA_{8,16}$)
RCA_F, RCA_G	FR-PreMatch (RCA_{32} + $ELMA_{8,16}$)
RCA_H	PreMatch ($RCA_{32,16,8,4,2}$)
RCA_I	FR-PreMatch (RCA_{32} + $minpower_{16-8-4-2}$)
$CSkA_A$	Base $CSkA_{32}$
$CSkA_B$, $CSkA_C$	PreMatch ($CSkA_{32}$ + $CSkA_{8,16}$)
$CSkA_D$, $CSkA_E$	FR-PreMatch ($CSkA_{32}$ + $ELMA_{8,16}$)
$CSkA_F$, $CSkA_G$	FR-PreMatch ($CSkA_{32}$ + $RCA_{8,16}$)
$CSkA_H$	PreMatch ($CSkA_{32,16,8,4,2}$)
$CSkA_I$	FR-PreMatch ($CSkA_{32}$ + $minpower_{16-8-4-2}$)
$ELMA_A$	Base $ELMA_{32}$
$ELMA_B$, $ELMA_C$	PreMatch ($ELMA_{32}$ + $ELMA_{8,16}$)
$ELMA_D$, $ELMA_E$	FR-PreMatch ($ELMA_{32}$ + $CSkA_{8,16}$)
$ELMA_F$, $ELMA_G$	FR-PreMatch ($ELMA_{32}$ + $RCA_{8,16}$)
$ELMA_H$	PreMatch ($ELMA_{32,16,8,4,2}$)
$ELMA_I$	FR-PreMatch ($ELMA_{32}$ + $minpower_{16-8-4-2}$)

RCA_n:	Ripple-Carry Adder (n-bits)
$CSkA_n$:	Carry Skip Adder (n-bits)
$ELMA_n$:	ELM Adder (n-bits)

TABLE II
BASE PROCESSOR PARAMETERS.

Base Processor Parameters	
Processor Speed	1GHz
Fetch/Retire Rate	1 per cycle
Functional Units	4 Int, 4 FP, 2 Address generators
Integer FU Latencies	1/7/12 add/multiply/divide (pipelined)
FP FU Latencies	4 default, 12 div. (all but div. pipelined)
Instruction Window	128 entries
Memory queue size	32 entries
Branch Prediction	2KB bimodal agree, 32 entry RAS
Base Memory Hierarchy Parameters	
L1 (Data)	64KB, 2-way associative, 64B line, 2 ports, 12 MSHRs
L1 (Instr)	16KB, direct mapped
L2 (Unified)	1MB, 4-way associative, 64B line, 1 port, 12 MSHRs
Main Memory	16B/cycle, 4-way interleaved
Base Contentionless Memory Latencies	
L1 (Data) hit time (on-chip)	2 cycles
L2 hit time (off-chip)	20 cycles
Main Memory (off-chip)	102 cycles

For this study we used the reference inputs for the SPECint2000 suite. However due to the long execution times of such inputs we first warm up the architecture and then simulate a 100 million instruction window using a more detailed simulator.

B. Results

In this section we analyze the energy consumption of the machines applying PreMatch and both PreMatch and FR (different hardware algorithms for different bitwidths). Although the results show absolute values over the execution of the benchmarks the analysis should be based on their relative values.

During our preliminary study on the effective bitwidth on the benchmarks we observed some benchmarks have very distinct patterns on the distribution of the bitwidths which directly influence the total energy consumption consumed during the execution of the benchmarks. In Fig. 4 this specificity of effective bitwidth on the benchmarks can be observe in all three major types of machines (RCA, CSkA and ELMA machines). The *minpowerbw1,bw2* means the cores which present minimum power consumption at bitwidths *bw1* and *bw2*, independent of their specific implementation.

For the RCA machines (Fig. 4(a)), comparing to the base machine (RCA_A), the biggest energy savings in average for PreMatch types (RCA_B, RCA_C and RCA_H) were 19.2%, 17.7% and 26% respectively; FR-PreMatch types present a maximum 26.6% for RCA_I. The same configurations present a maximum energy reduction of 29.6% for `cc1` benchmark and 23.9%, 35.3% and 35.8% for `twolf`.

TABLE III
ENERGY PARAM. FOR THE WATTCH TOOLSET.

FU type	Energy consumption [nJ]	Relative # of cycles	Aver. Energy per clock [nJ]
$ELMA_{32}$	28.493	1	28.493
$ELMA_{16}$	12.297	1	12.297
$ELMA_8$	5.021	1	5.021
$CSkA_{32}$	23.317	3	7.772
$CSkA_{16}$	11.316	2	5.658
$CSkA_8$	5.495	1	5.495
RCA_{32}	19.611	5	3.922
RCA_{16}	9.796	3	3.265
RCA_8	4.878	2	2.439

For the CSkA machines (Fig. 4(b)), it was possible to get from 19.6% to 26.5% on PreMatch-only machines and as much as 28.0% using FR and PreMatch together.

In the ELMA machines (Fig. 4(c)), we observed the major savings: 21.1%, 20.1% and 28.4% in average for $ELMA_B$, $ELMA_C$ and $ELMA_H$ types. $ELMA_I$ showed -29.3% energy comparing to the base machine. This same machine presented a 39.5% energy saving for `twolf`.

FR and PreMatch together in average showed short performance improvement comparing to PreMatch only, except when using circuits for all precisions (I-type machines). They presented in average 0.4% to 1.5% improvements over the only PreMatch counterparts (type H machines). One should notice however those are machines which use most silicon area and this factor should not be neglected when evaluating those machines. Fig. 4(d)) shows how much energy can be saved without any performance impact (PreMatch).

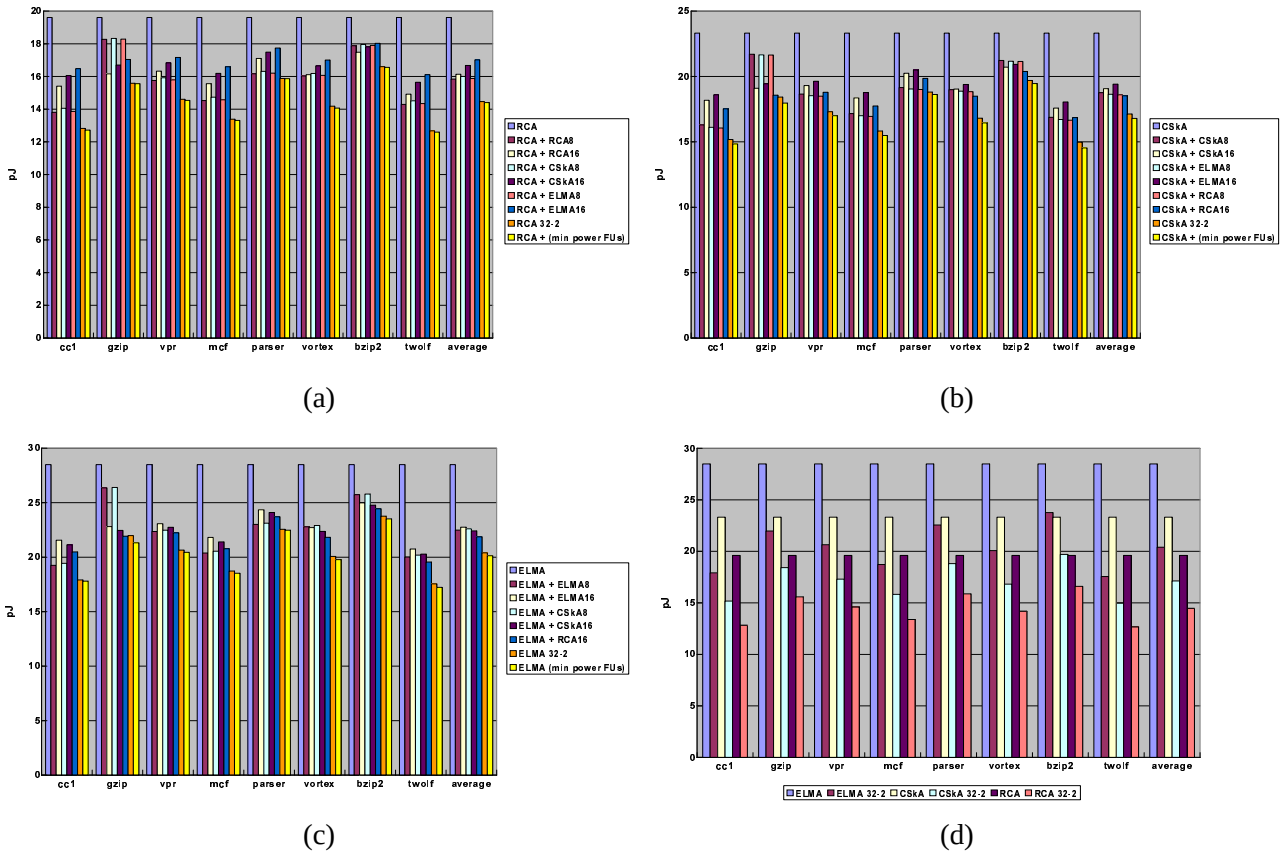

Fig. 4. Energy per addition in (a) RCA-machines; (b) CSkA-machines; (c) ELMA-machines. In (d) is the detail on PreMatch machines exploring all precisions (32-16-8-4-2).

All machines presented great reductions in energy consumption specially with `cc1` and `twolf`, as the percentage of narrow-operands is very high 68.8% (up to 8-bits) for the first and 47.8% (up to 16-bits) for the other. Another interesting result is `gzip` which optimizes very well for configurations which include a 16-bit redundant core (C, E, G, H and I types).

V. Related Work and Discussion

Exploitation of subword operations in order to improve processor power and performance has been subject of recent investigation. Subword operations are called "narrow width" in [3]. In this paper they propose two methods to exploit narrow width data to improve power consumption and performance: (a) a clock gating scheme, which disables the upper bits of the operands based on their values; (b) packing multiple narrow operations into a single ALU (a type of SIMD).

The reductions observed on the first scheme are due to clock gating applied based on operands' values for a 64-bit Alpha-like processor datapath. Clock gating is also applied in our scheme. Although they detect bitwidths dynamically, the matching to the effective precision is done at design time, i.e., which subword precision has to be implemented in conjunction with the zero-detect circuit **for each FU**. A system with a relatively wide datapath with many FU's would have a not so negligible power and area overhead due to the zero-detect and clock gating control circuits. A great number of subword precisions for each adder is also impractical as the overhead of the decision logic would increase the critical path of the adder.

In our approach, as the zero-detect circuit and the clock-gating control are independent and not implemented in each FU, the impact in area and power is smaller for the same number of FU's. Another reason for this is that one more redundant module using PreMatch does not necessarily introduce the double of interconnections and wires. For example, in a 64-bit datapath with 4 FU's of 64-bits and one increases the number of FU's with 4 8-bit FU's more, the total increase in interconnections and control logic are 12.5% for the buses and more 1-bit for the control logic which now needs a 3-to-8 decoder instead of a 2-to-4 for the operands.

Our proposed scheme though uses the effective bitwidth

of each operation, which can also be applied in bitwidth-aware systems, i.e., a variable designed to have a dynamic range of 10-bits can still benefit from PreMatch if its effective values range under this range. A number of works have showed this is the common case and that the dynamic range of variables varies from application to application and also with the input data. Providing the system with a number of circuits with some specific precisions leads to a simpler design and consequently, shorter TAT.

However providing too many narrow-width modules would impact performance and power consumption due to the increase in complexity of the control unit and the number of multiplexers and interconnections necessary. This tradeoff has to be thoroughly quantified according to the number and types of redundant precision-aware FU's, the effective precisions to be supported (according to the application data-width pattern), which impacts the number or interconnections and the coding/decoding circuit, and as a whole the overhead on leakage power. However the major percentage of leakage power is due to the on-chip caches and memories as over 80% of the chip area is consumed by caches/memories in modern processors, making the area and leakage power percentage overhead introduced by redundant cores small. A more precise analysis on the tradeoff between the number of redundant cores, its granularity and their impact in the processor implementation is scheduled as future work.

In this paper we were interested in the dynamic power reductions that can be obtained using the proposed techniques. We assume unused cores are clock gated or have their power supply cut-off to reduce leakage power consumption. The interconnect overhead is limited to the execution stage of the pipeline when the operands are dispatched to the FU's which are going to perform the computations.

In our work we try to analyze the benefits of subword operations independently from the already known benefits of clock gating [6]. Our mechanism employs similar principle for power reduction to precomputation, guarded evaluation or signal gating: reduce useless switching on circuits. However, those techniques do not take advantage of different power consumption, computation time and parallelism inherent to FR systems; moreover, the precision matching capability is only limited by the number and types of redundant modules what is not easily fulfilled with the other approaches.

VI. Summary and Future Directions

We proposed a low-power data-driven dynamic module selection technique for Functional Redundant systems called **PreMatch—Effective Precision Matching Computation**. We argue that the execution of arithmetic instructions may waste less power or be completed faster if executed by a different circuit which approximately match the precision dictated by the contents of its operands (effective bitwidth) during execution.

We demonstrated the effectiveness and feasibility of *Functional Redundancy (FR)* and *Effective Precision Matching Computation (PreMatch)* as techniques to reduce energy consumption and/or improve computation time of arithmetic modules. Module selection based on the computation effective precision showed that up to 28.4% power savings can be achieved with PreMatch alone, and 29.3% using both techniques (FR and PreMatch). The ammount of power saved with FR and PreMatch is heavily dependent on the power and performance characteristics of the different implementations for a given precision; the use of FR permits to use different hardware algorithms which would be impractical at some precision bitwidth due to power or area constraints (e.g. Wallace tree multipliers depending on the system may be impractical for 32-bits, but their speed could be used for 16- or 8-bit precision multiplications). Even bigger power savings are expected for wider datapaths.

We show up to 29.3% energy savings can be obtained on `add` instructions only. PreMatch can easily be applied to other functions like multiplication or even more complex functions. These present greater room for power savings and/or performance improvements by use of faster algorithms for small operands[1].

This research aims at minimizing the energy per clock with low or none performance overhead. PreMatch presents no overhead on performance however FR does. This leads to a new problem, the module selection for FR systems. We are actually working on heuristics to this problem and ways to explore the slack of instructions in order to obtain even more energy savings in order to extend the battery-life of mobile systems.

References

[1] O. J. Bedrij. Carry-select adder. In *IRE Transaction on Electronic Computers*, volume 11, pages 340–346. June 1962.

[2] R. P. Brent and H. T. Kung. A regular layout for parallel adders. *IEEE Trans. Computers*, C-31(3):260–264, 1982.

[3] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proc. of HPCA*.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proc. of The 28th ISCA*, pages 83–94, 2000.

[1] Normally high performance algorithms, which are disregarded at design time due to their complexity or excessive silicon area, may now be used for smaller operands

[5] Y. Cao and H. Yasuura. A system-level energy minimization using datapath optimization. In *Proc. of ISLPED*.

[6] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, September 1996.

[7] V. M. Goulart Ferreira and H. Yasuura. Functional redundancy for dynamic exploitation of performance-energy comsumption trade-offs. In *Proc. of the 13^{th} Symposium on Integrated Circuits and Systems Design*, pages 165–170, September 2000.

[8] J. L. Hennessy and D. A. Patterson. *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., second edition, 1990.

[9] T. P. Kelliher, R. M. Owens, M. J. Irwin, and T.-T. Hwang. ELM—a fast addition algorithm discovered by a program. In *IEEE Trans. on Computers*, volume 41. September 1992.

[10] C. Nagendra, M. J. Irwin, and R. M. Owens. Area-time-power tradeoffs in parallel adders. *IEEE Trans. CAS-II*, 43(10):689–702, October 1996.

[11] R. Pyreddy and G. Tyson. Evaluating design tradeoffs in dual speed pipelines. In *Proc. of the Workshop of Complexity-Effective Design (ISCA'2001)*. David Albonesi, June 2001.

[12] M. Stephenson, J. Babb, and S. Amarasinghe. Bitwidth analysis with application to silicon compilation. In *Proc. of PLDI*.

[13] V. Tiwari et al. Reducing power in high-performance microprocessors. In *Proc. of the 35^{th} ACM/IEEE Design Automation Conference (DAC)*, June 1998.

[14] S. Turrini. Optimal group distribution in carry-skip adders. In *Proc. of 9^{th} IEEE Symp. on Computer Arithmetic*, pages 96–103, 1989.